NLP 243 Assignment 1

Shubham Gaur
UC Santa Cruz
sgaur2@ucsc.edu

1 Introduction

The problem addressed in this project is a supervised multi-label classification task. The goal is to predict multiple attributes associated with movie descriptions based on textual input data. Each movie can be linked to multiple attributes such as genres, actors, directors, and other relevant features. Therefore, the classification problem requires the model to identify all relevant labels for a given input instance (movie description) rather than just one. This specific task involves:
Input: Textual descriptions of movies.
Output: A set of labels indicating the attributes of the movie, represented as a binary vector where each element corresponds to a specific attribute.

2 Descriptive Statistics

Background Information
The dataset provided consists of two main files: a training dataset (hw1_train.csv) and a test dataset (hw1_test.csv). The training dataset includes various movie descriptions along with their associated attributes. This dataset is critical for training a multi-label classification model that will generalize well to unseen movie descriptions.

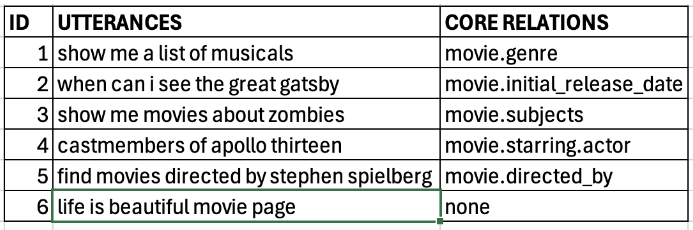
Training Data: hw1_train.csv
Number of samples: 2312
Number of features: 2 (input attributes)
Number of labels: 19 (output attributes)
Number of None values: 319

Test Data: hw1_test.csv
Number of samples: 981
Number of features: 1 (input attributes)

3 Input and Output

Input

- **Column:** UTTERANCES



ID	UTTERANCES	CORE RELATIONS
1	show me a list of musicals	movie.genre
2	when can i see the great gatsby	movie.initial_release_date
3	show me movies about zombies	movie.subjects
4	castmembers of apollo thirteen	movie.starring.actor
5	find movies directed by stephen spielberg	movie.directed_by
6	life is beautiful movie page	none

Figure 1: Example showcasing the input and output, including cases with "none":

- **Description:** This column contains the textual description of each movie. It may include plot summaries, keywords, or other relevant details that help identify the movie attributes.

- **Output**
  - **Column:** CORE RELATIONS
  - **Description:** This column contains a space-separated string of attributes associated with each movie description. Each movie can have multiple attributes, and a "none" value is used to indicate that no attributes apply.

4 Models

4.1 Embedding Methods

In our project, I used a character-level embedding technique called Count Vectorizer with n-grams to effectively capture the intricacies of the movie descriptions we are working with.

4.2 Count Vectorizer (Character-Level N-grams)

- **What It Is:** The Count Vectorizer is a powerful tool that converts text into a matrix of character counts. By focusing on character sequences (n-grams), we can capture not just words but the very structure of the text itself.

- **Why I Chose It:** Movie descriptions can be quite diverse and may contain spelling variations, typos, or unique character combinations. By using character-level embeddings, we can handle these variations more robustly. This method helps us capture subtle patterns that might be overlooked if we were to focus solely on words.

- **How It Works:** We configured the Count Vectorizer to analyze characters in sequences ranging from single characters to five-character combinations. This flexibility allows our model to learn from a wide range of character patterns.

4.3 Multi Layer Perceptron

The MLP (Multi-Layer Perceptron Clevert et al. (2015)) model is at the heart of our classification task. It's a straightforward yet powerful neural network architecture tailored for multi-label classification.

4.4 Implementation

Architecture Overview

- The model starts with an input layer that accepts the character-level embeddings I've generated.

- It has three hidden layers (512, 256, and 128 neurons), each using ReLU (Rectified Linear Unit) activation, which helps introduce non-linearity to the model.

- I've added dropout layers after each hidden layer to keep our model from overfitting, which is a common issue when dealing with complex datasets.

- Finally, the output layer employs a sigmoid activation function to give us probabilities for each label, making it perfect for multi-label classification tasks.

Training

- **The Learning Process:** We trained our model using Binary Cross-Entropy Loss, which is suitable for our multi-label problem. We chose the Adam optimizer for its efficiency and adaptive learning capabilities.

- **Keeping Track:** We implemented a learning rate scheduler that reduces the learning rate when the validation loss plateaus, ensuring our model keeps improving over time.

Tunable Hyperparameters

- **Learning Rate:** Set at 0.0005, which helps the model learn at a controlled pace.

- **Dropout Rate:** Set at 0.2 to avoid overfitting, ensuring that the model generalizes well on unseen data.

- **Batch Size:** We use a batch size of 64, which is a sweet spot for balancing memory usage and training speed.

- **Number of Epochs:** Initially set to 60, with a mechanism to stop early if no improvements are seen on the validation set.

Evaluation Metrics

- We evaluate our model's performance using accuracy, and we also plot precision-recall curves for each class to visualize how well our model performs at various thresholds.

Citations

- The architecture of our neural network draws inspiration from foundational work in the field of deep learning, particularly from:
  - Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*.
  - Zhang, Y., Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Proceedings of the 2015 Conference on *Empirical Methods in Natural Language Processing*.

5 Experiments

5.1 Dataset Split

In this project, we utilized a dataset consisting of movie descriptions and their corresponding labels, which indicate various attributes related to the movies. The dataset was split into training and validation sets to evaluate the model's performance effectively.

- **Training Set:** 90

- **Validation Set:** 10

The split was performed using train_test_split from the sklearn.model_selection library, ensuring that the distribution of labels remained consistent across both sets.

5.2 Hyperparameter Selection

To optimize the performance of our models, we conducted a systematic search for hyperparameters. This included testing various learning rates, batch sizes, dropout rates, and the number of hidden units in the neural networks. We utilized techniques such as grid search and random search over a defined parameter space to identify the best-performing configurations.

5.3 Model Approaches

Baseline - Word2Vec with FastText Embedding

Description: This approach utilized Word2Vec along with FastText embeddings, which provide word vectors based on character n-grams. FastText captures subword information, making it robust against out-of-vocabulary words. This technique is especially valuable for dealing with domain-specific terms or rare words in movie descriptions.

Embedding Method

- **Dataset:** I used the crawl-300d-2M-subword.vec dataset, which contains 2 million word vectors trained on the Common Crawl dataset. This extensive dataset allows us to leverage rich semantic representations of words and their meanings.

- **Window Size:** Adjustments were made to the context window size to see how it affected the capture of word semantics.

- **Embedding Size:** This parameter was set to discard less frequent words, optimizing the vocabulary size and improving model focus on relevant terms.

Configuration:

- **Hidden Layers:** The architecture features two hidden layers with 128 and 64 units. This setup helps the model learn complex relationships in the data.

- **Batch Size:** I chose a batch size of 32. This strikes a good balance between the stability of our gradient estimates and the speed of training.

- **Learning Rate:** I set the learning rate to 0.001, which encourages gradual convergence during training.

- **Weight Decay:** A weight decay of 0.0001 is applied to help prevent overfitting by penalizing overly large weights.

- **Training Process:**

  Loss Function: During training, I utilized binary cross-entropy loss, which is well-suited for multi-label classification tasks.

  Training Performance:

  – **Training Loss:** The training loss consistently decreases, indicating that the model is learning effectively. The validation loss shows some fluctuations but generally trends downward, suggesting the model is improving in generalizing from the training data, despite a bit of instability.
  – **Validation Accuracy:** The training accuracy soars close to 100%, which raises a red flag for potential overfitting. Meanwhile, validation accuracy stabilizes around 0.8, indicating there's room for improvement in how well the model generalizes to unseen data.

  Threshold Setting:

  – **Chosen Threshold:** I decided to set the threshold at 0.2 for binarizing the model's outputs based on a careful analysis of the Precision-Recall curve.
  – **Implications:** This lenient threshold classifies more samples as positive, which is useful in a multi-label classification context. However, I will monitor precision and recall metrics to manage false positives.

  Performance Constraints:

  – **Runtime Limitations:** A key challenge was ensuring the entire application runs within a 10-minute window. Loading the FastText .vec file alone takes over 7 minutes, affecting real-time usability.
  – **Optimization Considerations:** To address this, I'm considering optimizations such as using smaller embedding files,

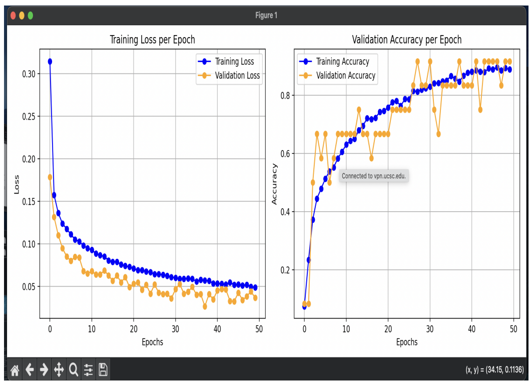


Figure 2: Training Loss, Validation loss along with training and validation accuracy - Baseline Model

improving data loading, and implementing caching strategies to enhance performance in future runs.

Conclusion:

- The Word2Vec model with FastText embedding achieved a training accuracy of 84% and a validation accuracy of 83%. However, upon submission on Kaggle Competition, the model's accuracy on the test set dropped to 74%, indicating potential overfitting.
- This discrepancy suggests that while the model performed well on the training and validation datasets, it struggled to generalize to unseen data, possibly due to differences in data distribution or characteristics.

5.4 Final Model - CountVectorizer

CountVectorizer is a key tool in natural language processing (NLP) that converts text data into a matrix of token counts, facilitating machine learning applications. Its ability to analyze text through n-grams—sequences of items (words or characters)—enhances the model's understanding of language context.

Feature Engineering

I implemented CountVectorizer with a hybrid character analyzer, specifically using analyzer='char_wb' and setting the ngram_range to (1, 5). This configuration allows us to capture a rich set of character-based features by generating n-grams that respect word boundaries.

Dataset Split

The dataset was split into training and validation sets to allow for effective model evaluation. Specifically, we applied a 90/10 split, where 80% of the data was used for training the model, while the remaining 20% served as the validation set. This increased exposure can enhance the model's ability to learn the underlying patterns, leading to better performance on unseen data.

Hyperparameter Selection

- **Learning Rate:** Various learning rates (e.g., 0.001, 0.0005) were evaluated to determine the optimal rate that yields the best convergence speed without causing instability during training.
- **Batch Size:** We experimented with different batch sizes (e.g., 32, 64) to balance the trade-off between training time and model performance.
- **Dropout Rate:** Different dropout rates (e.g., 0.1, 0.2) were tested to reduce overfitting by randomly deactivating a portion of neurons during training.
- **Hidden Layer Sizes:** The architecture of the neural network was adjusted by changing the sizes of hidden layers, specifically testing configurations like (512, 256, 128, 64).

Handling Data Sparsity and Imbalance

- **CountVectorizer with Hybrid Analyzer:** We utilized a CountVectorizer with the hybrid analyzer set to char_wb, which allowed us to capture character-level n-grams. This approach is particularly effective in dealing with rare words and improving representation for underrepresented classes.
- **Oversampling/Undersampling Techniques:** Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or random undersampling were considered to balance class distributions, although this was not implemented in the final model due to time constraints.

Optimisation Techniques

- **Learning Rate Scheduler:** I utilized the Adam optimizer with an initial learning rate of 0.0005, known for its efficiency in handling sparse gradients. To optimize training, I initially applied the ReduceLROnPlateau scheduler, which reduces the learning rate by a factor of 0.1 when the validation loss stagnates.

However, this approach did not yield significant improvements, leading me to adopt a fixed learning rate of 0.005 to ensure more stable and effective training. I also Optuna for optimisation and found the best learning rate to be 0004753560317202933(0.0005).

- **Weight Decay:** I incorporated weight decay (L2 regularization) to prevent overfitting by penalizing large weights in the model (was getting 10% difference in the training and the validation accuracy) The optimal weight decay parameter was determined using **Optuna**, an automatic hyperparameter optimization framework was 1.273979973319028e-05 (1e-5). This approach involved training over 100 samples to find the best combination of weight decay and learning rate, enhancing the generalization of the model.

- **Early Stopping:** The model was overfitting, so I implemented early stopping, with a patience paramater value of 10, meaning if no improvement in the 10 consecutive epoch passes, then it considers that epoch as the convergence point. I found 45 epochs to be the appropriate number of passes to train the data.

- **Batch Normalization:** To stabilize the learning process and accelerate convergence, Batch Normalization was applied after each linear layer. This technique normalizes the outputs of a previous activation layer by adjusting and scaling the activations.

- **Dropout Regularization:** To reduce overfitting, we used Dropout layers in our model architecture. Dropout randomly sets a fraction of the input units to zero during training, I used a dropout value of 0.2.

- **Precision-Recall Optimization:** I calculated optimal thresholds for each class by analyzing the precision-recall curve. For each class, I extracted precision, recall, and thresholds using the precision_recall_curve function. Subsequently, I computed the F1 scores, which provide a balance between precision and recall, and identified the threshold that maximizes the F1 score. This approach ensures that the selected threshold effectively balances false positives and false negatives for each class, improving model performance on the validation set. The optimal thresholds are stored for further evaluation and predictions.

Table 1: Classification Report

Class	**Precision**	**Recall**	**F1-score**
actor.gender	0.00	0.00	0.00
gr.amount	0.00	0.00	0.00
movie.country	0.95	0.95	0.95
movie.directed_by	0.92	0.92	0.92
movie.estimated_budget	1.00	0.87	0.93
movie.genre	1.00	0.89	0.94
movie.gross_revenue	0.86	0.75	0.80
movie.initial_release_date	0.97	0.95	0.96
movie.language	1.00	0.97	0.99
movie.locations	0.00	0.00	0.00
movie.music	0.00	0.00	0.00
movie.produced_by	0.89	1.00	0.94
movie.production_companies	0.86	0.86	0.86
movie.rating	1.00	0.95	0.98
movie.starring.actor	0.92	0.95	0.94
movie.starring.character	1.00	1.00	1.00
movie.subjects	1.00	0.90	0.95
none	0.93	0.89	0.91
person.date_of_birth	0.00	0.00	0.00
micro avg	0.95	0.92	0.93
macro avg	0.70	0.68	0.69
weighted avg	0.94	0.92	0.93
samples avg	0.92	0.92	0.92

For each class, the F1 score was calculated as the harmonic mean of precision and recall. The optimal threshold for each class was identified using the argmax function, which selects the index corresponding to the maximum F1 score. **The average threshold has been calculated and stands at 0.38.**

6 Results

The performance of the model was evaluated on the training and validation datasets over 45 epochs. The key statistics recorded are as follows:

Training Loss: 0.0070
Validation Loss: 0.02777
Training Accuracy: 0.9993
Validation Accuracy: 0.891

Training and Validation Loss
The training loss decreased significantly over the epochs, reflecting the model's effective learning process. In contrast, the validation loss showed minor fluctuations but remained consistently low, indicating the model's ability to generalize from the training data.

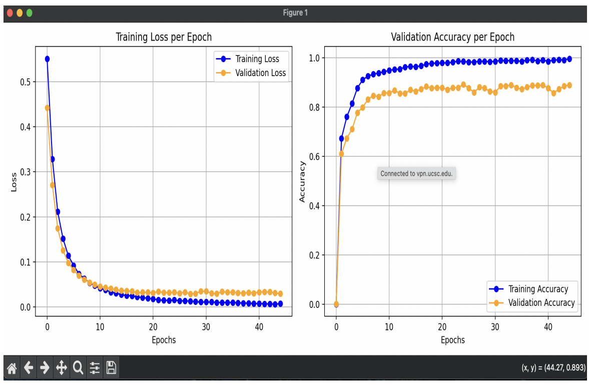


Figure 3: Training & Validation - Loss & Accuracy

Visual Representation Figure 3 provides a visual representation of the training loss and validation loss across the epochs. The blue curve illustrates the training loss, which steadily decreases, while the orange curve represents the validation loss, which stabilizes after a brief fluctuation.

7 Conclusion

In this study, a classification model utilizing CountVectorizer with n-grams was implemented to predict various attributes in movie datasets. The model demonstrated robust performance, achieving an impressive accuracy of **81.855% on the test set, placing it eighth on the leaderboard**. The results indicate the effectiveness of the chosen embedding and modeling techniques, highlighting the potential of n-gram features to improve classification tasks in natural language processing. Future work may explore further optimizations and additional feature engineering to improve predictive performance even further.

References

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). In *Proceedings of the International Conference on Learning Representations (ICLR)*.